RECEIVED
CENTRAL FAX CENTER

AUG 0 1 2005

PATENT
Atty. Dkt. No. LCNT/120651
Serial No. 09/815,942

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:
Stephen Blott et al, Appellant

Serial No.: 09/815,942

Confirmation No.: 8249

Filed: 3/23/2001

For: APPLICATION PROGRAMMING INTERFACE FOR NETWORK APPLICATIONS

§

Group Art Unit: 2154

Examiner: Siddiqi, Mohammad A.

MAIL STOP APPEAL BRIEF-PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATE OF MAILING OR TRANSMISSION
I hereby certify that this correspondence is being deposited on with the United States Postal Service as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, or being facsimile transmitted to the USPTO, on the date indicated below.

8/1/05
Date

Signature LAURA E CRATER

Dear Sir:

APPEAL BRIEF

Appellant submits this Appeal Brief to the Board of Patent Appeals and Interferences on appeal from the decision of the Examiner of Group Art Unit 2154 dated March 24, 2005, finally rejecting claims 1-20. The final rejection of claims 1-20 is appealed. This Appeal Brief is believed to be timely filed by the due date of August 1, 2005, as set by mailing a Notice of Appeal on June 1, 2005. The Commissioner is authorized to charge the $500 Appeal Brief filing fee, and any additional fees required to make this Appeal Brief timely and acceptable to the Office, to counsel's Deposit Account No. 20-0782/LCNT/120651.

RECEIVED
OIPE/IAP

AUG 0 8 2005

08/08/2005 JBALINAN 00000029 200782 09815942

01 FC:1402 500.00 DA

TABLE OF CONTENTS

1.	Identification Page	1
2.	Table of Contents	2
3.	Real Party in Interest	3
4.	Related Appeals and Interferences	4
5.	Status of Claims	5
6.	Status of Amendments	6
7.	Summary of Claimed Subject Matter	7
8.	Grounds of Rejection to be Reviewed on Appeal	11
9.	Arguments	12
10.	Conclusion	18
11.	Claims Appendix	19
12.	Evidence Appendix	24
13.	Related Proceedings Appendix	25

Real Party in Interest

The present application has been assigned to Lucent Technologies Inc. of Murray Hill, New Jersey.

Related Appeals and Interferences

Appellant asserts that no other appeals or interferences are known to Appellant, Appellant's legal representative, or assignee, which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

Status of Claims

Claims 1-20 are pending in the application. Claims 1-20 were originally filed in the application. Claims 1-20 stand finally rejected as discussed below. The final rejections of claims 1-20 are appealed. The pending claims are shown in the attached Claims Appendix.

Status of Amendments

All claim amendments have been entered by the Examiner.

Summary of Claimed Subject Matter

The embodiments of the present invention generally are directed to an application programming interface (API) for network applications processing packets having associated with them source nodes and destination nodes. An application programming interface according to the invention allows creating or destroying one or more pairs of data structures for asynchronously passing between the operating system and the network application pointers to packet buffers mapped to both the operating system and the network application. The network application may use the pair to input packets received from a specified network interface and output packets to be processed by the operating system's network layer as received packets. The network application may also use the pair to input packets provided by the operating system's network layer to a specified network interface, and output packets to be sent via that interface. The network application may also use the pair to input packets received and output packets to be sent by a specified network interface. The network application may also use such a pair to input allocated and output deallocated packet buffers.

Generally speaking, the invention utilizes packet buffer data structures to share data between a network application and an operating system. Within the context of the present invention, as incoming packets arrive, the packets are stored in memory. Data structures are created wherein pointers are placed in the data structures by the operating system. Specifically, pointers that point to the location of the incoming packets in memory are placed in a detour queue 104A and taken out of the detour queue 104A by a network application 108. After the packets have been processed by the network application 108, the pointers are placed in revert queue 106A by network application 108 and taken out via operating system 102 for transmission out to the network via a respective device such as a host application 110, TCP/IP implementation 112 or directly to the plurality of network interfaces 114. Thus, the processing or exchanging of pointers by both the operating system and the network application enables an ability to process associated packets by the OS and network application.

Of interest within the context of the present invention: (1) all packet buffers are mapped both to the system and to network applications; (2) instead of passing copies of packets to one another, system and network applications exchange pointers to packets;

and (3) system and network applications communicate asynchronously via a number of circular queues, thus avoiding system call overheads in normal cases.

An application programming interface (API) according to an embodiment of the invention is described in detail in FIG. 1 and its accompanying text (paragraphs 23-29 which may be found on pages 7-9).

In the embodiments of independent claim 1, an application programming interface (API) for network applications capable of processing packets having source and destination node addresses different from a node where the application runs comprises:

> first (104A, 104B) and second (106A, 106B) data structures associated with a network interface (114) in communication with a network (116), said first and second data structures being mapped (mbuf_map 208) to an operating system (102) and a network application (108), said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein:
>
> packets to be passed from the operating system to the network application are stored in a buffer (402) and referenced via respective pointers (410) within said first data structure (104A, 104B), said respective pointers being exchanged between said operating system and said network application, said first data structure pointers being inserted (204) into said first data structure by said operating system prior to network layer processing, said first data structures being removed (206) by said network application, insertion and removal of said first data structure pointers being asynchronous with respect to each other; and
>
> packets to be processed as received packets by said network layer of said operating system are stored in a buffer (402) and referenced via respective pointers (410) within said second data structure (106A, 106B), said respective pointers being exchanged between said network application and said operating system, said second data structure pointers being inserted into said second data structure by said network application, said second data structure pointers being removed by said operating system, insertion and removal of said second data structure pointers being asynchronous with respect to each other.

In the embodiments of independent claim 17, an application programming interface (API) for network applications, which applications can process packets whose source and destination node addresses are nodes different from a node where the application runs, said API comprising:

a primitive (208) for creating a first (104A, 104B) and a second (106A, 106B) data structures associated with a specified network interface (114), if said data structures do not exist, and mapping said data structures both to the operating system (102) and a specified network application (108), said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein

the specified network interface receives and sends packets from and to a network,

each said packet is stored in a buffer (402) mapped both to the operating system and the specified network application,

the operating system inserts into and the specified network application removes from said first data structure, a pointer (410) to each buffer (402) containing a packet that the operating system's network layer outputs to the specified network interface, before the network interface sends said packets, said insertions and removals being asynchronous with respect to each other, and

the specified network application inserts into and the operating system removes from said second data structure, a pointer to each buffer containing a packet that the specified network interface sends to the network, said insertions and removals being asynchronous with respect to each other.

In the embodiments of claim 19, an application programming interface (API) for network application, which applications can process packets whose source and destination node addresses are nodes different from a node where the application runs, said API comprising:

a primitive (208) for creating a first (104A, 104B) and a second (106A, 106B) data structures associated with a specified network interface (114), if said data structures do not exist, and mapping said data structures both to the operating system (102) and a specified network application (108), said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein

the specified network interface receives and sends packets from and to a network and does not require a coprocessor,

the specified network application requires supervisor privileges, every packet is stored in a buffer mapped both to the operating system and every network application,

the operating system's network and higher protocol layers do not process any packets that the specified network interface receives or sends.

the operating system inserts into and the specified network application removes from said first data structure, a pointer to each buffer containing a packet that the specified network interface receives from the network, said insertions and removals being asynchronous with respect to each other, and

the specified network application inserts into and the operating system removes from said second data structure, a pointer to each buffer containing a packet that the specified network interface sends to the network, said insertions and removals being asynchronous with respect to each other.

Grounds of Rejection to be Reviewed on Appeal

Claims 1-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Mahler et al. (6,675,218) (hereinafter Mahler) in view of Garcia et al.(6,145,061) (hereinafter Garcia).

ARGUMENTS

Obviousness of Claims 1-20 over Mahler in view of Garcia

Claims 1-20 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Mahler in view of Garcia.

The Mahler reference discloses a system for user-space network packet modification. The Mahler reference defines a packet modification library (PML) and associated machine language instructions to implement the PML in kernel space and user space. A PML application programming interface (API) provides functions to a programmer such that packet taps may be set up and packets may be moved between kernel space and user space (column 6, lines 15-32).

A PML kernel "open" function allocates space in the kernel for a "packet queue" to hold pointers to trapped packets, where the trapped packets are logically viewed as a "file." Thereafter, when PML application code seeks to "read" from that "file," a read thread created by PML kernel code will wait until a packet pointer is present in the packet queue and will then move the associated trapped packet into user space and delete the packet from kernel space. Similarly, when PML application code seeks to "write" a packet to the "file," PML kernel code will move the packet into kernel space, and PML application code will delete the packet from user space (column 7, lines 18-29). Thus, trapped packets are moved between kernel space and application space via a file read or write operation.

The Garcia arrangement provides a method of management of a circular queue for asynchronous access. Specifically, two separate processing elements operate to asynchronously access and manage a circular queue. A specific methodology is defined whereby an order of data storage and removal is provided, which order is assisted by adding to the queue (along with each data element) a zero data element that marks the tail of the queue and signifies that the queue is empty.

Appellant's invention of claim 1 recites the following:

> An application programming interface (API) for network applications capable of processing packets having source and destination node

addresses different from a node where the application runs, said API comprising:

first and second data structures associated with a network interface in communication with a network, said first and second data structures being mapped to an operating system and a network application, said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein:

packets to be passed from the operating system to the network application are stored in a buffer and referenced via respective pointers within said first data structure, said respective pointers being exchanged between said operating system and said network application, said first data structure pointers being inserted into said first data structure by said operating system prior to network layer processing, said first data structure pointers being removed by said network application, insertion and removal of said first data structure pointers being asynchronous with respect to each other; and

packets to be processed as received packets by said network layer of said operating system are stored in a buffer and referenced via respective pointers within said second data structure, said respective pointers being exchanged between said network application and said operating system, said second data structure pointers being inserted into said second data structure by said network application, said second data structure pointers being removed by said operating system, insertion and removal of said second data structure pointers being asynchronous with respect to each other.

In contrast to the above-quoted claim language, the Mahler reference fails to teach or disclose at least the claimed "said respective pointers being exchanged between said operating system and said network application ...." Specifically, the claimed invention contemplates "packets to be passed from the operating system to the network application" and "packets to be processed as received packets by the network layer of said operating system ...." In each instance, the packets are "stored in the buffer and referenced via respective pointers [wherein] said respective pointers being exchanged between said operating system and said network application...."

To the extent that the Mahler reference utilizes pointers, those pointers are only used to point to a specific PML kernel function or point from a "packet queue" to trapped packets. A packet read or packet write operation according to Mahler comprises utilizing a pointer to copy a pointed-to packet or write a packet to a pointed-to location, respectively. That is, pointers are used within the normal context of a file read/file write

operation to retrieve or place data in a file. A file read operation returns to the calling program a packet while a file write operation is passed a packet(s) by the calling program. More particular, there is no application-level utilization of the pointers, and certainly no exchanging of pointers between an operating system and an application, as claimed.

Thus, the Mahler reference fails to disclose or suggest the claimed invention. It is noted that the significant gap between the claimed invention and the Mahler reference is simply not bridged by the Garcia reference. The Garcia reference provides disclosure related to a circular queue for asynchronous access. There is no teaching or suggestion within Garcia of the claimed "exchanging of pointers" as noted above with respect to claim 1. Thus, at least the exchanging of pointers claim element is missing from the claimed references, either singly or in any combination.

Appellant does not agree with the Examiner's position and analysis of Appellant's various responses. The Examiner's attempt to clarify the alleged teachings of Mahler in, illustratively, the Final Office Action are deficient; thus, any attempted combination of Mahler with other references still does not result in the claimed invention. Independent Claims 1, 17 and 19 recited and relevant case law cited in Appellant's prior response are still of record and are referred to hereby by Appellant.

Specifically, in Section 23 of the Final Office Action, the Examiner stated that Mahler discloses respective pointers being exchanged between said operating system and said network by the application running in user space and the PML API moving a packet between kernel space and user space per FIG. 1, Col. 6, lines -32, Col. 2, line 36 – Col. 3, line 11. However, in addition to the above discussion, it is noted that FIG. 1 (per Col 6, lines 1-3) shows a block diagram of the interoperation and division of a TCP/IP protocol stack between user space and kernel space, no specific indication of pointer usage between such memory regions is depicted. Additionally, Col. 2, line 36 – Col. 3, line 11 specifically discusses the copying of packets to move such packets from kernel space to user space and vice versa. This in no way discloses usage of pointers in accordance with the subject invention and is actually a teaching counter thereto. That is, if the actual packets are manipulated (i.e., copied) then there is no suggestion of the implementation (or exchange) of pointers as claimed. Also, Col. 6, lines (15)-32

specifically discloses the moving of packets between kernel space and user space (lines 27-28). As stated earlier, this is inopposite to the purpose and proposed solution of the subject invention. Accordingly, there can be no teaching or suggestion of pointer usage as claimed.

The Examiner believes that Mahler discloses each said packet is stored in a buffer mapped both to the operating system and the specified network application (via the file-desc pointer of Col 6, lines 63-67 and Col. 7, lines 1-6). However, this portion of the reference teaches pointer usage in association with function calls (i.e., read, write, open and the like) of applications in user space in relation to kernel space and not the specific mapping or referencing of packets as claimed in the subject invention. At Col 6, line 65 – Col 7. line 3, "...the kernel allocates memory in kernel space for a "file structure," which will hold pointers to the beginning and end of the file on a disk, as well as pointers to the kernel functions that implement the "close", "read", "write" and "ioctl" user-space functions. In addition, the kernel returns to the calling application a pointer "file-desc" to the file structure. It is respectfully submitted that there is no discussion of the packets in the cited section because it was not the intention of Mahler to map packets but functions via pointers. Furthermore, subsequent manipulation of packets based on this pointer activity was cited by the Examiner in the previous Office Action (Col. 7, lines 7-29) and such was shown to not be combinable with other cited references to result in the claimed invention in Appellant's prior response. That is, the combination would merely disclose a kernel of an operating system allocating memory in kernel space for a file structure which will hold pointers to the beginning and the end of the file on a disk, allocating memory in the kernel space for a packet queue to hold pointers to trap packets, moving the associated trapped packet into user space and deleting the packet from the kernel space, and using the queue for storing data elements so that it can be managed asynchronously by separate processing elements.

As such, Appellant submits that independent claims 1, 17 and 19 are not obvious and fully satisfy the requirements of 35 U.S.C. §103 and is patentable thereunder. Furthermore, claims 2-16, 18 and 20 depend directly or indirectly from independent claims 1, 17 and 19 and recite additional limitations thereof. As such and at least for the same reasons as discussed above, Appellant submits that these dependent claims are

also not obvious and fully satisfy the requirements of 35 U.S.C. §103 and are patentable thereunder. Therefore, Appellant respectfully requests that the Examiner's rejection be withdrawn.

Independent claim 17 (and similarly independent claim 19) recites:

> "An application programming interface (API) for network applications, which applications can process packets whose source and destination node addresses are nodes different from a node where the application runs, said API comprising:
>
> a primitive for creating a first and a second data structures associated with a specified network interface, if said data structures do not exist, and mapping said data structures both to the operating system and a specified network application, said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein
>
> the specified network interface receives and sends packets from and to a network,
>
> each said packet is stored in a buffer mapped both to the operating system and the specified network application,
>
> the operating system inserts into and the specified network application removes from said first data structure, a pointer to each buffer containing a packet that the operating system's network layer outputs to the specified network interface, before the network interface sends said packets, said insertions and removals being asynchronous with respect to each other, and
>
> the specified network application inserts into and the operating system removes from said second data structure, a pointer to each buffer containing a packet that the specified network interface sends to the network, said insertions and removals being asynchronous with respect to each other." (emphasis added)

As discussed above, the combination of Mahler and Garcia discloses passing packets between the kernel space and the user space and using the queue for storing data elements so that it can be asynchronously managed by separate processing elements. Nowhere in the combined references is there any teaching or suggestion of "each said packet is stored in a buffer mapped both to the operating system and the specified network application." That is, Appellant's invention stores the packet in a buffer that is mapped both to the operating system and the specified network application. Instead of passing copies of packets between the operating system and the network application, the kernel and user space exchange pointers to the packet (see

Appellant's specification, page 9, paragraph 30). Thus, Appellant's invention enhances performance over the cited references, since it eliminates the copying of the packets from the kernel space to the user space, and vice versa. Therefore, since the cited references fail to teach or suggest "each said packet is stored in a buffer mapped both to the operating system and the specified network application," the combined references fail to teach or suggest Appellant's invention as a whole.

As such, Appellant submits that independent claims 17 and 19 are not obvious and fully satisfy the requirements of 35 U.S.C. §103 and are patentable thereunder. Furthermore, claims 18 and 20 depend directly or indirectly from independent claims 17 and 19 and recite additional limitations thereof. As such and at least for the same reasons as discussed above, Appellant submits that these dependent claims are also not obvious and fully satisfy the requirements of 35 U.S.C. §103 and are patentable thereunder.

CONCLUSION

Thus, Appellant submits that none of the claims presently in the application are obvious under the provisions of 35 U.S.C. §103. Consequently, Appellant believes all these claims are presently in condition for allowance.

For the reasons advanced above, Appellant respectfully urges that the rejections of claims 1-20 as being obvious under 35 U.S.C. §103 are improper. Reversal of the rejections of the Final Office Action is respectfully requested.

Respectfully submitted,

Eamon J. Wall
Registration No. 39,414
Moser, Patterson & Sheridan, L.L.P.
595 Shrewsbury Avenue, Suite 100
Shrewsbury, New Jersey 07702
Telephone: 732-530-9404
Telephone: 732-530-9808

CLAIMS APPENDIX

1. (Previously presented) An application programming interface (API) for network applications capable of processing packets having source and destination node addresses different from a node where the application runs, said API comprising:

first and second data structures associated with a network interface in communication with a network, said first and second data structures being mapped to an operating system and a network application, said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein:

packets to be passed from the operating system to the network application are stored in a buffer and referenced via respective pointers within said first data structure, said respective pointers being exchanged between said operating system and said network application, said first data structure pointers being inserted into said first data structure by said operating system prior to network layer processing, said first data structure pointers being removed by said network application, insertion and removal of said first data structure pointers being asynchronous with respect to each other; and

packets to be processed as received packets by said network layer of said operating system are stored in a buffer and referenced via respective pointers within said second data structure, said respective pointers being exchanged between said network application and said operating system, said second data structure pointers being inserted into said second data structure by said network application, said second data structure pointers being removed by said operating system, insertion and removal of said second data structure pointers being asynchronous with respect to each other.

2. (Original) The API of claim 1, further comprising a primitive for creating said first and a second data structures if said first and a second data structures are not available.

3. (Original) The API of claim 1, further comprising a primitive for unmapping said first and a second data structures from the network application, said unmapping primitive operating to destroy said first and a second data structures if said data structures are mapped to no other network application.

4. (Original) The API of claim 3, wherein:

in the case of said first and a second data structures not being associated with the network interface, packets to be passed between the network and the network interface are processed by the operating system network layer.

5. (Original) The API of claim 1, wherein the operating system's network layer implements the Internet Protocol (IP).

6. (Original) The API of claim 1, further comprising a primitive for creating said first and a second data structures mapped both to said operating system and said network application, wherein:

non-network packets to be passed from the operating system to the network application are stored in a buffer and referenced via respective pointers within said first data structure, said first data structure pointers being inserted into said first data structure by said operating system, said first data structure pointers being removed by said network application; and

non-network packets to be passed from said network application to said operating system are stored in a buffer and referenced via respective pointers within said second data structure, said second data structure pointers being inserted into said second data structure by said network application, said second data structure pointers being removed by said operating system.

7. (Original) The API of claim 6, wherein the operating system maintains in said first data structure at least a predefined number of pointers.

8. (Original) The API of claim 6, wherein the API further comprises a primitive to destroy said first and second data structures.

9. (Original) The API of claim 1 wherein other network applications are prevented from accessing a buffer from the time said network application removes a pointer to said buffer from said first data structure and inserts a pointer to said buffer into said second data structure.

10. (Original) The API of claim 9, wherein each buffer contains an identifier of any network application having exclusive use of the buffer.

11. (Original) The API of claim 10, wherein upon termination of a network application, the operating system automatically reclaims buffers that are in the application's exclusive use.

12. (Original) The API of claim 1 wherein said first or second data structure is a circular queue.

13. (Original) The API of claim 1, further comprising a primitive for placing the network application in a quiescent state until the operating system inserts a pointer into said first data structure.

14. (Original) The API of claim 1, further comprising a primitive for placing the network application in a quiescent state until the operating system removes a pointer from said second data structure.

15. (Original) The API of claim 1, wherein the node where the network application runs is configured as one of a host, a bridge, a switch and a router.

16. (Original) The API of claim 6 wherein other network applications are prevented from accessing a buffer from the time said network application removes a pointer to said

buffer from said first data structure and inserts a pointer to said buffer into said second data structure.

17. (Previously presented) An application programming interface (API) for network applications, which applications can process packets whose source and destination node addresses are nodes different from a node where the application runs, said API comprising:

a primitive for creating a first and a second data structures associated with a specified network interface, if said data structures do not exist, and mapping said data structures both to the operating system and a specified network application, said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein

the specified network interface receives and sends packets from and to a network,

each said packet is stored in a buffer mapped both to the operating system and the specified network application,

the operating system inserts into and the specified network application removes from said first data structure, a pointer to each buffer containing a packet that the operating system's network layer outputs to the specified network interface, before the network interface sends said packets, said insertions and removals being asynchronous with respect to each other, and

the specified network application inserts into and the operating system removes from said second data structure, a pointer to each buffer containing a packet that the specified network interface sends to the network, said insertions and removals being asynchronous with respect to each other.

18. (Original) The API of claim 17, wherein the API further comprises a primitive for unmapping said data structures from the specified network application and, if said data structures are mapped to no other network application, destroying said data structures.

19. (Previously presented) An application programming interface (API) for network application, which applications can process packets whose source and destination node addresses are nodes different from a node where the application runs, said API comprising:

a primitive for creating a first and a second data structures associated with a specified network interface, if said data structures do not exist, and mapping said data structures both to the operating system and a specified network application, said network interface, operating system, and network application residing at a node capable of processing packets having source and destination node addresses different from said node where the application runs, wherein

the specified network interface receives and sends packets from and to a network and does not require a coprocessor,

the specified network application requires supervisor privileges,

every packet is stored in a buffer mapped both to the operating system and every network application,

the operating system's network and higher protocol layers do not process any packets that the specified network interface receives or sends,

the operating system inserts into and the specified network application removes from said first data structure, a pointer to each buffer containing a packet that the specified network interface receives from the network, said insertions and removals being asynchronous with respect to each other, and

the specified network application inserts into and the operating system removes from said second data structure, a pointer to each buffer containing a packet that the specified network interface sends to the network, said insertions and removals being asynchronous with respect to each other.

20. (Original) The API of claim 19, wherein the API further comprises a primitive for unmapping said data structures from the specified network application and, if said data structures are mapped to no other network application, destroying said data structures.

EVIDENCE APPENDIX

None

THIS PAGE BLANK (USPTO)

RELATED PROCEEDINGS APPENDIX

None

THIS PAGE BLANK (USPTO)